



AFRL-OSR-VA-TR-2013-0204

Architectural Support for Detection and Recovery using Hardware Wrappers

**Bhagirath Narahari
Rahul Simha**

The George Washington University

**April 2013
Final Report**

DISTRIBUTION A: Approved for public release.

**AIR FORCE RESEARCH LABORATORY
AF OFFICE OF SCIENTIFIC RESEARCH (AFOSR)
ARLINGTON, VIRGINIA 22203
AIR FORCE MATERIEL COMMAND**

REPORT DOCUMENTATION PAGE
*Form Approved
OMB No. 0704-0188*

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to the Department of Defense, Executive Services and Communications Directorate (0704-0188). Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE (DD-MM-YYYY) 02-26-2013	2. REPORT TYPE FINAL REPORT	3. DATES COVERED (From - To) March 1, 2009 - Nov 30, 2012		
4. TITLE AND SUBTITLE Architectural Support for Detection and Recovery using Hardware Wrappers		5a. CONTRACT NUMBER		
		5b. GRANT NUMBER FA9550-09-1-0194		
		5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Narahari, Bhagirath Simha, Rahul		5d. PROJECT NUMBER		
		5e. TASK NUMBER		
		5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) The George Washington University 2121 I St, NW, 6th Floor Washington, DC 20052		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Office of Scientific Research 875 N. Randolph St Room 3112 Arlington VA 22203		10. SPONSOR/MONITOR'S ACRONYM(S)		
		11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-OSR-VA-TR-2013-0204		
12. DISTRIBUTION/AVAILABILITY STATEMENT DISTRIBUTION A: APPROVED FOR PUBLIC RELEASE				
13. SUPPLEMENTARY NOTES				
14. ABSTRACT The objective of this project was the development of secure execution environments for applications that use third party software components developed by a variety of vendors, and restrict how code shares the application memory space, and provide isolation within the application space. A hardware-software approach was taken to provide fine grained memory access protection by placing each software component or package in a hardware wrapper which enforces limits on the resources accessed by these software packages, and thus helps detect an attack and enables recovery from an attack. Current computing platforms were augmented with hardware that enforces limits on resources accessed by the software packages – these hardware wrappers constrain the damage that can be done by a malicious software package and maintain a stable system through recovery mechanisms. Extensive experiments, which revealed modest performance overhead, conducted on a full system simulation infrastructure demonstrated that fine grained memory protection using the concept of wrappers is both practical and effective.				
15. SUBJECT TERMS security, hardware architectures, software security, authorization, memory protection,				
16. SECURITY CLASSIFICATION OF: a. REPORT U		17. LIMITATION OF ABSTRACT SAR	18. NUMBER OF PAGES 15	19a. NAME OF RESPONSIBLE PERSON Bhagirath Narahari
				19b. TELEPHONE NUMBER (Include area code) 202 994 8323

*Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18*

Final Performance Report

Architectural Support for Detection and Recovery using Hardware Wrappers

AFOSR Contract Number FA9550-09-1-0194

Principal Investigator:

Bhagirath Narahari

Department of Computer Science
The George Washington University
2121 I St. NW, Suite 601
Washington, DC 20052
narahari@gwu.edu
(202)-994-8323

Cover Sheet: Form SF 298 attached.

1. Executive Summary.

The overall objective of the project was to enable the development and execution of secure applications that use third party software components developed by a variety of vendors, and restrict how code shares the application memory space, and provide isolation within the application space. An integrated hardware-software approach was developed, and this has the potential of significantly raising the barrier against such attacks and that, in addition, includes an orderly recovery process. The solution augments current computing platforms with hardware that enforces limits on resources accessed by the software packages – these hardware *wrappers* ensure that a software application can only read its own memory, cannot engage in denial-of-service and that a violation results in a hardware-supervised automated recovery process. The hardware wrapper is based on a manifest generated during application development process and contains information on authorized memory access policies, timing information, and recovery code. The conventional CPU is modified to incorporate an enforcement engine which checks the manifest at run-time to detect and prevent any unauthorized access to memory and to invoke recovery process after an attack is detected. In addition to the original objectives, investigations were started on topics related to the project objectives with a focus on embedded systems. Modern embedded control systems feature multiple processors that must coordinate sensing and action. For this reason, the research explored extending the monitoring and recovery features to the context of distributed control. In particular, the focus was on language and runtime support for distributed control of actuators in the presence of attacks and failures.

To achieve the goal, research was conducted to integrate and advance current techniques in security, computer architecture, compilers, and languages to develop novel system to protect against attacks through backdoors placed in third party software. The specific research objectives included:

- Architecture design to implement the hardware wrappers and containers concept
- Manifest/meta-information needed to track authorized accesses
- Testbed and simulation infrastructure development to quantify and measure performance in terms of architecture parameters.
- Performance analysis to evaluate overhead incurred by our approach,
- Design, and evaluation, of language and runtime support for distributed control of actuators in the presence of attacks and failures.

The innovation in the approach was in developing a solution that exploited the power of integrated software-hardware techniques. The results of this research included architectures for fine grained protection of memory and involved design of algorithms, models for automated recovery, programming language and runtime support, architecture design, compiler techniques, simulations, and performance analysis. Several techniques were proposed and designed to address these objectives. The research involved faculty and graduate students at the doctoral level, and partly supported two completed doctoral dissertations. The overall impact of the research is in the development of secure applications that use third party code.

2. Status of Effort:

The research proposed involved a number of research areas: (1) design of the architecture, and compiler support, for protection against attacks on an application through backdoors placed in third party code; (2) explore the use of reconfigurable logic as a hardware acceleration platform to implement security mechanisms; (3) exploring system support for hardware structures; and (4) development of complete architecture and system simulation infrastructure to measure the effectiveness of the proposed solution.

A hardware-software approach was taken to provide a secure execution environment that uses third party code. The idea is to prevent attacks launched from third party code through application's memory space. The solution approach places each software package in a hardware wrapper, or each software component in a hardware container. (We use the terms wrapper and container to primarily distinguish the granularity at which we apply our software protection technique.) The innovation in our approach is to augment the standard von Neumann architecture with a few hardware primitives to carefully check memory accesses and CPU usage at runtime so that no application, or software package, can access memory beyond specific bounds nor can slow down the CPU by hogging compute-time. In addition, we performed preliminary research that focused on language and runtime support for secure embedded system in the context of a distributed control system.

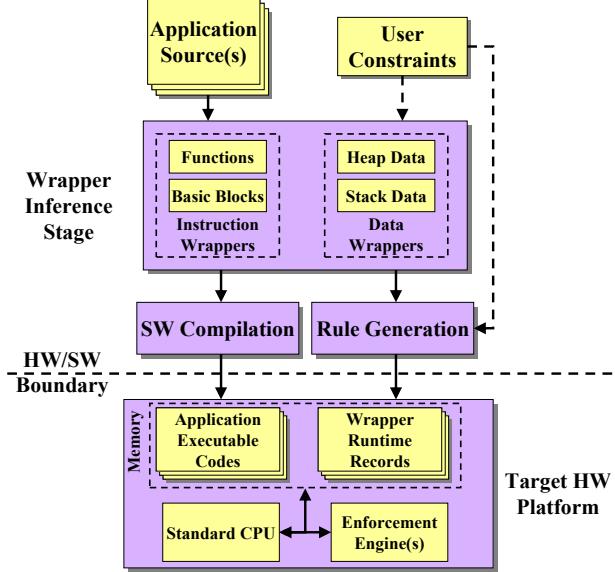


Figure 1: Hardware-Software Processes in our Approach

The effort for developing our secure execution environment lies on two fronts, as shown in Figure 1, which illustrates the hardware-software processes in our approach:

- (a) On the software engineering front are extraction of program properties (i.e., the meta properties needed to track accesses), compiler techniques, and writing of recovery code that satisfies the developer’s security policies. These properties would determine the manifest/meta-information needed to track unauthorized access. In addition, on the secure, robust embedded systems with distributed control, the development of language and runtime features for distributed control of actuators in the presence of attacks, failures, and timing constraints.
- (b) On the computer architecture front, design of a trusted execution platform that validates fine-grained memory access, checks control flow, and enforce an orderly recovery procedure. In addition, design of high-throughput, low-latency cryptographic hardware using reconfigurable logic to provide fast implementation of cryptographic techniques.

We demonstrated through a series of experiments that fine-grained memory protection, as provided by our approach, is both practical and achievable. Further, we showed that architectural optimizations and careful design and integration of wrappers with the processor pipeline could result in modest performance penalty while providing improved security. This suggests that our approach of hardware wrappers provides an effective secure execution environment for modern embedded systems which use third party code.

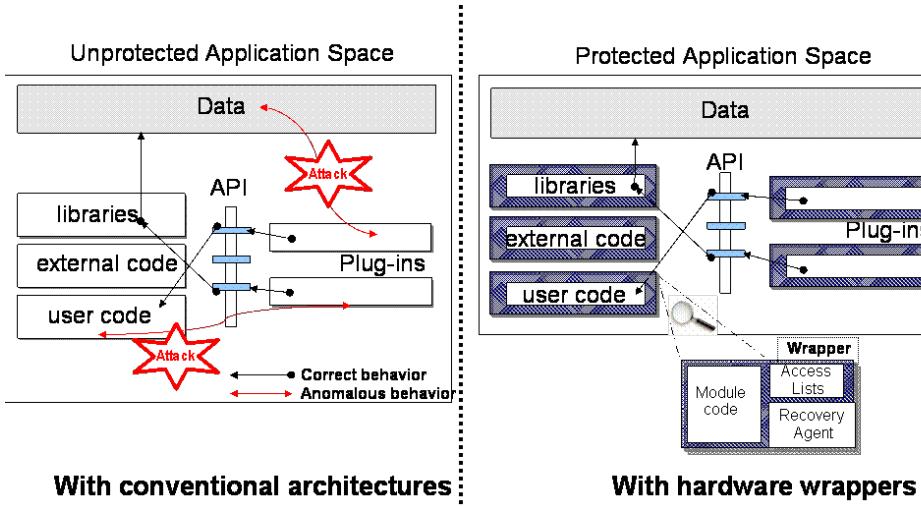


Figure 2: Wrapper Concept

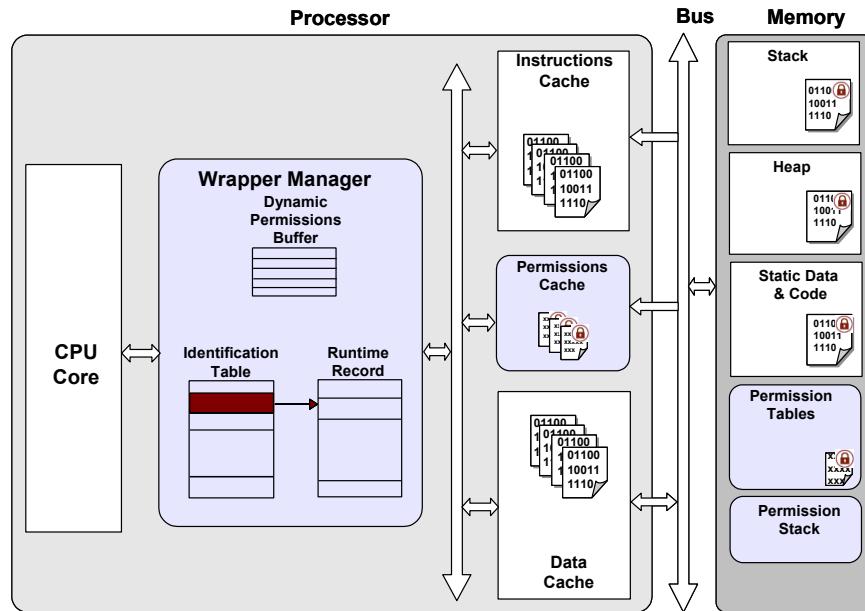


Figure 3: Architecture Overview

3. Summary of Accomplishments

We made progress on several fronts in the project, both on the primary objective of developing the hardware wrappers concept as well as other complementary objectives that were identified during our investigations. Below we itemize these achievements by

topic and summarize the main results obtained. More detailed results are available in the papers listed in the publications section, and available at <http://www.seas.gwu.edu/~narahari/afosr/> .

1. Hardware Wrapper Concept and Architecture:

An application may be developed using open source third party software, including libraries and plug-ins. In the conventional architecture, shown on the left half of Figure 1, malicious code in the third-party software can make an unauthorized access to the user code's memory (since the user specified this malicious code as part of their application) and launch various kinds of attacks on the code and data. In our approach [15,16], as shown in the right half of the Figure 2, a conventional application's components are instrumented with a *wrapper*, which is a *manifest* containing metadata created at the time of application development that specifies the resources needed by individual software components/packages and also contains recovery code. Our approach modifies the standard von Neumann architecture, Figure 3, by providing a wrapper enforcement engine to enable fast runtime checking of memory accesses and CPU usage by packages. These checks are made against the *manifest*. At run-time, the hardware detects improper memory accesses (reads or writes) or denials-of-service (slowing down the CPU) and can detect an attack or initiate a hardware-supervised recovery process that restores operation and also records a snapshot of events that might assist a forensic examination of the attack. Thus, a malicious component that misuses a pointer, to make an unauthorized memory access, or runs in an infinite loop, will be trapped by the supporting hardware and the attack will be detected and prevented. The hardware can then oversee an orderly recovery. The recovery process we outlined in [17] could itself be a chain of interrupts going back up the component creation hierarchy all of which is supervised and enforced by the hardware so that the recovery process itself cannot be hijacked.

A primary goal of our Hardware Wrappers approach was to make both our hardware and software enhancements as transparent as possible. In the case of hardware, we provide the bulk of the needed hardware functionality in a module called the *Wrapper Manager* that sits between the CPU and cache as shown in the Figure 3. In this manner, a processor designer would merely insert our module and interface it with both the CPU and the cache. Similarly, our goal in software is to have most of the work done by the compiler and loader, with minimal work required for the programmer. In fact, the only action on the part of the programmer is to define the units of isolation – this requires modification of a software development tool that builds the application (such as *make*). Figure 3 also shows the additional bookkeeping data (the permissions) stored in main memory, which are fetched into the Wrapper Manager as needed to enforce isolation boundaries.

The wrapper architecture, a hardware-software technique to provide fine-grained memory protection, was evaluated by through a prototype implementation and measuring the performance overhead. Several techniques were developed to reduce the performance overhead. We built a full system support for fine-grained memory access and developed of a full-blown system simulator using Opal/Gems toolkit. Further, we explored concurrency support for management of wrappers and evaluating and improving

multithreaded performance by optimizing cost of context switches. Extensive experiments were conducted using our simulation infrastructure, and the salient experimental results are tabulated in Appendix A along with details of the experimental platform. Some key accomplishments and observations are summarized in what follows, and are reported in detail in [2,3,15,16].

- **Performance Overhead incurred by Hardware Wrappers.** The big challenge in our approach to fine-grained memory protection, using wrappers, is to minimize the performance overhead incurred by the extra cycles taken by the processor to check the authorizations, by checking the manifests in the wrapper enforcement engine. After careful architectural optimizations, our system incurred very modest average overhead of approximately 15% across a range of benchmarks. Figure 4 in Appendix provides the tabulated results of our experiments, including the effect of architectural optimizations on the performance.
- **Concurrency support for wrappers management.** The traditional model of processes and threads strongly relates to where protection boundaries exist. Instead of multiple execution sequences inside a protection domain, hardware wrappers introduce multiple protection domains inside a continuous execution sequence. Wrappers hardware automatically manages the context switch between security protection domains, but the hardware is unaware of a task context switch which requires loading the wrapper metadata for the new task. We extended RTEMS operating system with support for the new model of security context switching thus providing concurrency support in our wrapper architecture.
- **Evaluating and improving multithreaded performance.** The performance overhead caused by using hardware wrappers for multithreaded applications includes both the overhead of a single-threaded application and the increased cost of the context switch. We explored different strategies for optimizing context switches. By increasing bus width, between wrapper manager and cache, the overhead due to context switch was reduced to 8% (Figure 5).
- **Improving permission assignment for dynamic data structures.** Complex data structures, such as linked lists, trees or graphs, pose a challenge for permission assignments. In order to pass the full structure to another principal for processing the same processing as traversing the data structure is needed for security attributes assignment. We created a variant of the permission delegation primitive: ALLOWM (allow multi) that receives as parameters the number of ranges and the location in memory of security permission data structure that holds multiple delegation attributes. Such security attributes can be made part of the data structure itself and maintained by the base operations (such as add, remove, etc) and reused without re-parsing the structure.
- **A micro benchmark for evaluating wrappers.** We continued to enhance our benchmark suite with a set of synthetic tests that allow us to vary program features that affect the performance cost of using wrappers.
- **Enhancing the security and safety of object-oriented languages with wrappers.** Object-oriented programming languages like C++ allow developers to encapsulate the inner workings of object implementation with access specifiers

such as public and private. A compiler can then enforce encapsulation by checking whether object interactions violate the specified visibility. However, encapsulation is easily broken if a program uses direct memory access—for example, with pointers. Thus C++ programmers are taught that “private is not secure.” This maxim is a result of the general security problems that unchecked pointers can bring. Hardware wrappers enable fine-grained memory access control to constrain memory accesses to well-defined boundaries, which we use to provide programmers with the ability to enforce both encapsulation and memory safety for C++ programs. Providing applications written in C++ with internal memory protection—that is, protection between objects executing in the same thread—is difficult because of the intricate features in C++. In particular, inheritance and class composition frustrate hardware enforcement of encapsulation, and the dynamic behavior of objects at runtime, such as polymorphism and exception-handling, presents challenges for hardware monitoring of memory accesses.

2. Systems software support for hardware data structures.

As we explore the role of hardware and systems software in security we see that the interfaces between applications and memory are primitive and inefficient. For example, the ALLOWM primitive requires enhanced knowledge of how algorithms act on data structures. Using our simulation test bed we have developed novel techniques for managing data structures that admit to efficient hardware support [1,7]. In particular, we are using priority queues as an example data structure that has many practical applications—almost any sorting problem can be solved with a priority queue—and has an efficient hardware implementation. Our research explored how the limitations of the hardware (size and inflexibility) can be circumvented by systems software support. By moving data structures into hardware, we can implement tight memory access bounds on applications by constraining code to accessing data structures through hardware interfaces secured by hardware wrappers.

3. Untrusted Hardware: We also explored the complementary problem of designing a secure execution environment when the hardware, and specifically the Integrated Chip (IC), may be compromised at the fabrication foundry by third party developers. Our solutions explored (i) limiting the amount of data that could leak “out” of a hardware wrapper over a system memory bus [11], (ii) looking into some real-world implementation issues [14], and (iii) proposing mechanisms to validate that manufactured hardware is true to its design [8]. This is complementary to the main architectural wrappers approach and should provide consumers of our technology some added confidence that the wrapper hardware itself can be protected.

4. Language and Runtime Support for Distributed Control of Secure Embedded Systems. We explored the language support for embedded systems that feature distributed control, such as those applied in robotics, control software, and avionics. The goal was to understand the systems infrastructure needed to support robust, secure embedded systems. We explored an infrastructure for distributed control built on the Lua run-time system. We chose the Lua run-time system as it has a very small

code base, which made extending and modifying the language and run-time environment a fairly easy process. However, object-oriented programming in Lua is not as accessible as it is in other programming languages. To facilitate the ease of object-oriented programming in our system, we developed a fairly rudimentary Python to Lua translator. This translator will not convert legacy Python code to Lua, but any code written with the intent of being translated will work. We also ported Lua to RTEMS, which will allow us to modify the architectural simulator in the future to investigate how architectural modifications can better support the robustness of our system. Only a few changes were needed to get Lua to execute in RTEMS, and the main change was to Lua's timing mechanisms. Since RTEMS is run in a simulator, there is no macro defined which can be used to convert the `clock()` into seconds like Lua expects. A separate system call is used to access the ticks per second the simulator is configured to run. This change allows us to obtain fairly precise timing of not only program execution, but also specific language features we wish to test. Within the new Lua run-time system, we are implementing distributed control tasks that model multiple agents (e.g. robots) that must coordinate to complete each task. As each agent executes, its system-level parameters can be measured and maintained within a secure envelope defined by a wrapper.

5. Reconfigurable Logic for Cryptographic Needs of Wrapper Architecture.

We looked at the high-throughput, low-latency cryptographic needs of the wrapper hardware, based on its placement in the memory hierarchy [12,18,9,10]. A continued investigation into conventional private-key cryptography provided evidence that while throughput requirements could be met using conventional block ciphers such as AES, there would be (variable) high latency requirements. This could potentially cause problems for implementation in real-time systems. We then expanded our hardware crypto work to look at stream ciphers, and have found that approaches based on chaotic cryptography show some promise for this application [18]. With safety critical systems as the focus of our investigations, we also continued to explore solutions for hardening embedded platforms that operate in adversarial environments. This includes encrypted execution and control flow verification with minimum architectural changes [10].

5. Personnel Supported.

During the third year of the award:

At The George Washington University:

- Professor Bhagi Narahari (PI) supported in part during summer months.
- Professor Rahul Simha. Supported in part during summer months.
- Dr. Gedare Bloom, graduate (doctoral) student, graduated 2012. His thesis explored the use of hardware data structures.
- Dr. Eugen Leontie, graduate (doctoral) student, graduated 2012. His doctoral research was on the topic of using fine grained hardware wrappers – called containers – and is directly related to the objectives of this project.
- Professor Gabriel Palmer, Supported in part during summer months.

- Mr Christopher Smith, doctoral student, was partially supported by this grant. He explored language and runtime support for secure embedded systems.
- Mr. James Marshall, doctoral student, was partially supported by this grant. He explored distributed control task generation and simulation for secure embedded systems.

At Iowa State University:

- Professor Joseph Zambreño was partially supported in summer months.
- Dr Amit Pande, PhD 2010, was partially supported by this grant and worked on FPGA architectures.
- Alex Baumgarten, and Justin Rilling, were partially supported by this grant to help with implementing FPGA designs.

6. Publications:

1. G. Bloom, G. Parmer, B. Narahari, and R. Simha. Shared Hardware Data Structures for Hard Real-Time Systems, *12th International Conference on Embedded Software, EMSOFT 2012*, October 2012.
2. E. Leontie, G. Bloom, B. Narahari, and R. Simha. No Principal Too Small: Memory Access Control for Fine-Grained Protection Domains, *15th Euromicro Conference on Digital System Design, DSD 2012*, September 2012.
3. E. Leontie, G. Bloom, and R. Simha. Automation for Creating and Configuring Security Manifests for Hardware Containers, *4th Symposium on Configuration Analytics and Automation, SafeConfig 2011*, October 2011.
4. Stefan Popoveniuc, John Kelsey, Eugen Leontie, On the privacy threats of electronic poll books, *International Conference on Security and Cryptography SECRYPT 2011*, Seville, Spain 2011.
5. A. Baumgarten, M. Steffen, M. Clausman, and J. Zambreño. "A Case Study in Hardware Trojan Design and Implementation", *International Journal of Information Security (IJIS)*, vol. 10, no. 1, pp. 1-14, 2011.
6. Gedare Bloom, Eugen Leontie, Bhagirath Narahari, Rahul Simha, Hardware and Security: Vulnerabilities and Solutions, Book Chapter in "*Handbook on Securing Cyber-Physical Critical Infrastructure – Foundations and Challenges*", Elsevier, ISBN-13: 978-0124158153, 2011.
7. G. Bloom, G. Parmer, B. Narahari, and R. Simha. Real-Time Scheduling with Hardware Data Structures, Work-in-Progress Session, *IEEE Real-Time Systems Symposium, 2010. RTSS 2010*. December 2010.
8. G. Bloom, B. Narahari, and R. Simha. Fab Forensics: Increasing Trust in IC Fabrication, *IEEE International Conference on Technologies for Homeland Security, 2010. HST '10*. November 2010.
9. Eugen Leontie, Gedare Bloom, Olga Gelbart, Bhagirath Narahari, and Rahul Simha. "A Compiler-Hardware Technique for Protecting Against Buffer Overflow Attacks", *Journal of Information Assurance and Security (JIAS)*, vol. 5, no. 1, pp. 1-8, 2010.
10. Eugen Leontie, Olga Gelbart, Bhagirath Narahari, Rahul Simha, Detecting Memory Spoofing in Secure Embedded Systems using Cache-Aware FPGA

Guards, *Sixth International Conference on Information Assurance and Security, IAS2010*.

11. A. Das, G. Memik, J. Zambreño, and A. Choudhary. "Detecting/Preventing Information Leakage on the Memory Bus due to Malicious Hardware", *Proceedings of Design, Automation, and Test in Europe (DATE)*, March 2010.
12. A. Pande and J. Zambreño. "Efficient Mapping and Acceleration of AES on Custom Multi-Core Architectures", *Concurrency and Computation: Practice and Experience*, 2010.
13. Stefan Popoveniuc, Eugen Leontie, SAFE RPC - Auditing mixnets safely using Randomized Partial Checking, *International Conference on Security and Cryptography SECRIPT 2010*, Athens, Greece 2010
14. A. Baumgarten, A. Tyagi, and J. Zambreño. "Preventing Integrated Circuit Piracy using Reconfigurable Logic Barriers", *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 66-75, January 2010.
15. E. Leontie, G. Bloom, B. Narahari, R. Simha, and J. Zambreño. Hardware-enforced Fine-grained Isolation of Untrusted Code, *Proceedings of the CCS Workshop on Secure Execution of Untrusted Code (SecuCode)*, November 2009.
16. E. Leontie, G. Bloom, B. Narahari, R. Simha, and J. Zambreño. Hardware Containers for Software Components: A Trusted Platform for COTS-Based Systems, *2009 IEEE/IFIP International Symposium on Trusted Computing and Communications, TRUSTCOM 2009*, August 2009.
17. J. Sathre, A. Baumgarten, and J. Zambreño. "Architectural Support for Automated Software Attack Detection, Recovery, and Prevention", *Proceedings of the International Conference on Embedded and Ubiquitous Computing (EUC)*, August 2009.
18. A. Pande and J. Zambreño. "A Chaotic Encryption Scheme for Real-time Embedded Systems: Design and Implementation", *Telecommunication Systems*, 2011.

Doctoral Dissertations

1. Gedare Bloom. "Operating System Support for Shared Hardware Data Structures", Dissertation Thesis, The George Washington University, 2012.
2. Eugen Leontie. "Hardware-enforced Fine-grained Isolation of Untrusted Code", Dissertation Thesis, The George Washington University, 2012.

7. Interactions/Transitions:

Participation/Presentation at Meetings, Conferences, Seminars:

- Presentation of papers at various conferences.
- Bhagi Narahari, Integrated Hardware-Software approaches to software security, Indian Institute of Technology, Hyderabad, Dec.2010.
- Bhagi Narahari, Integrated Hardware-Software approaches to software security, Indian Institute of Technology, Bombay, Jan.2011.

- G. Bloom, *Hardware Data Structures*, Fifth Annual SEAS Student Research and Development Showcase, 2011. The George Washington University, Washington, DC
 - Meeting with Cisco groups at Bangalore, India to discuss hardware assisted security techniques (March 2012).

8. New Discoveries, inventions, patents: None.

9. Honors/Awards: N/A

Appendix A: Experimental Platform and Results.

We implemented and simulated our Wrapper architecture and evaluated its architectural effectiveness through extensive simulations.

Simulation Infrastructure and Test Bed.

In order to evaluate a system enhanced with our wrapper mechanism, we conducted a series of simulations that provide an accurate comparison between an unprotected system and a security enhanced device. As expected, the extra validations, code instrumentations and run-time metadata cause a performance penalty. We developed a simulator-based infrastructure for evaluating the performance impact of using wrappers to secure embedded systems. We implemented the wrapper hardware on top of a modern processor architecture based on the UltraSPARC III architecture. The UltraSPARC III represents an iteration of a long line of RISC processor designs, and it is equipped with state-of-the-art architectural features. Simics/GEMS implements a cycle-accurate model of a complex out-of-order architecture with functional simulation of the UltraSPARC III instruction set. We implemented extensions to GEMS and plugins for Simics to emulate the hardware features of the wrapper reference monitor. We chose architectural parameters of our simulation and experiments to match typical system-on-chip and embedded platforms available as commercial products. To demonstrate the feasibility of the fine-grained memory access control in a complex software environment we chose RTEMS as a suitable OS. We developed a Board Support Package (BSP) for the UltraSPARC III (and OpenSparc Niagara) and contributed it as the first 64-bit target port for RTEMS; it is now part of the upstream RTEMS distribution. RTEMS is POSIX-compliant and offers support for custom task extensions including a context switch call-out, which we utilized to implement the wrapper context switch. We evaluate the performance of our solution with experiments using benchmark applications from MiBench, the Data Intensive Systems (DIS) benchmark suite, a reduced size Dhrystone test, and the heap-intensive Richards benchmark. Graphical results are presented as the percent overhead compared with the baseline execution time, so a lower percentage represents better performance. Figure 4 shows the performance cost of using hardware wrappers across a sample of the benchmarks.

Experimental Results

- **Concurrency support for wrappers management.** The traditional model of processes and threads strongly relates to where protection boundaries exist. Instead of multiple execution sequences inside a protection domain, hardware wrappers introduce multiple protection domains inside a continuous execution sequence. Wrappers hardware automatically manages the context switch between security protection domains, but the hardware is unaware of a task context switch which requires loading the wrapper metadata for the new task. We extended RTEMS with support for the new model of security context switching: on a context switch, the OS saves together with the normal execution context (registers, program counter) the dynamic part of the security context (dynamic permission

buffer and dynamic permissions from the container run-time record) for the active container. We implemented the context switch as a RTEMS task extension. Task switching is triggered by the OS, which notifies the container manager by a dedicated instruction that handles security context saving, flushing and loading the new context before allowing the task to continue execution.

- **Evaluating and improving multithreaded performance.** The performance overhead caused by using hardware wrappers for multithreaded applications includes both the overhead of a single-threaded application and the increased cost of the context switch. We explore three strategies for optimizing context switches. The first is a simple extension to the controller logic that allows loads and stores to be validated against partially loaded permission list. If a hit occurs, the memory operation is valid. If not instead of triggering a security violation, the CM continues to stall the pipeline and retries the memory access once the permission list is fully loaded. Experimental results show us a reduction in stalled loads to as much as 21% and stalled stores up to 30% in the best cases. The second optimization considers a windowing mechanism that matches the register window of the processor for the container manager. Such a mechanism reduces many of the memory transfers during a security context switch, at the cost of a much higher chip space used. We obtained the most significant overall speedup after an architectural optimization—bus widening—that reduces the bandwidth bottleneck between the container manager and the permission cache. For register to memory operations a word size bus (16/32/64 bit) is typically used. For context switch operations, the CM transfers large continuous sequences, consisting of permission lists. Increasing the bus width by a factor of 2 or 4 significantly reduces the performance bottleneck. Figure 5 shows the cost of context switching as frequency increases, how adding wrapper management to the context switch increases that cost, and how bus widening reduces the cost.

- **A microbenchmark for evaluating wrappers.** We continued to enhance our benchmark suite with a set of synthetic tests that allow us to vary program features that affect the performance cost of using wrappers. In particular our microbenchmark allows us to control the rate and ratio of memory operations, function calls, and cpu-intensive workloads. Figure 6 shows the results of our synthetic tests as we vary the ratio of function calls to total number of instructions.

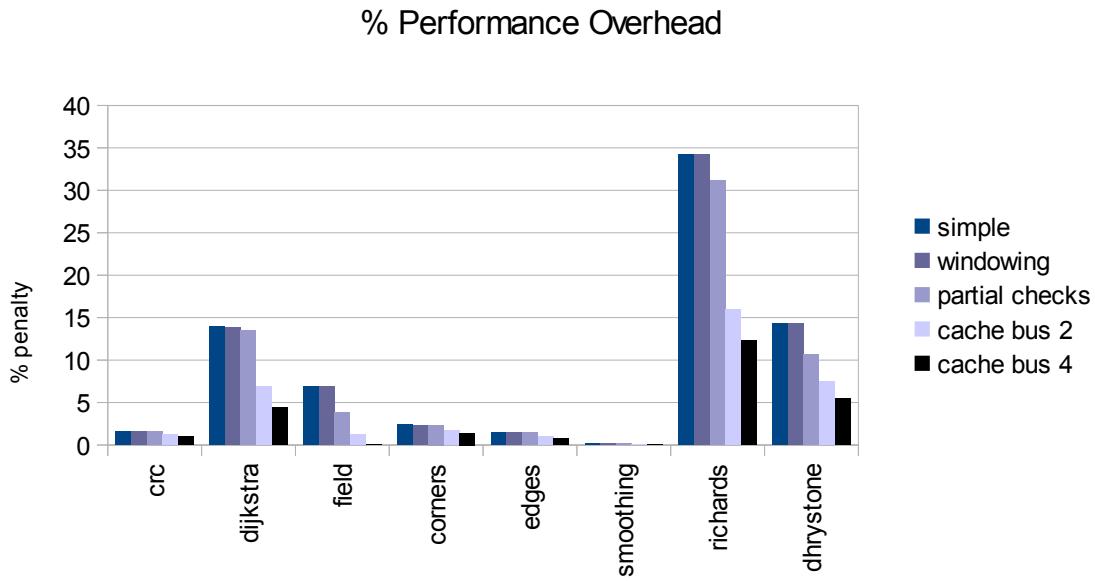


Figure 4. Performance cost of benchmarks with hardware wrappers (smaller is better). Hardware optimizations reduce the cost: most improvement is gained by increasing the cache bus width.

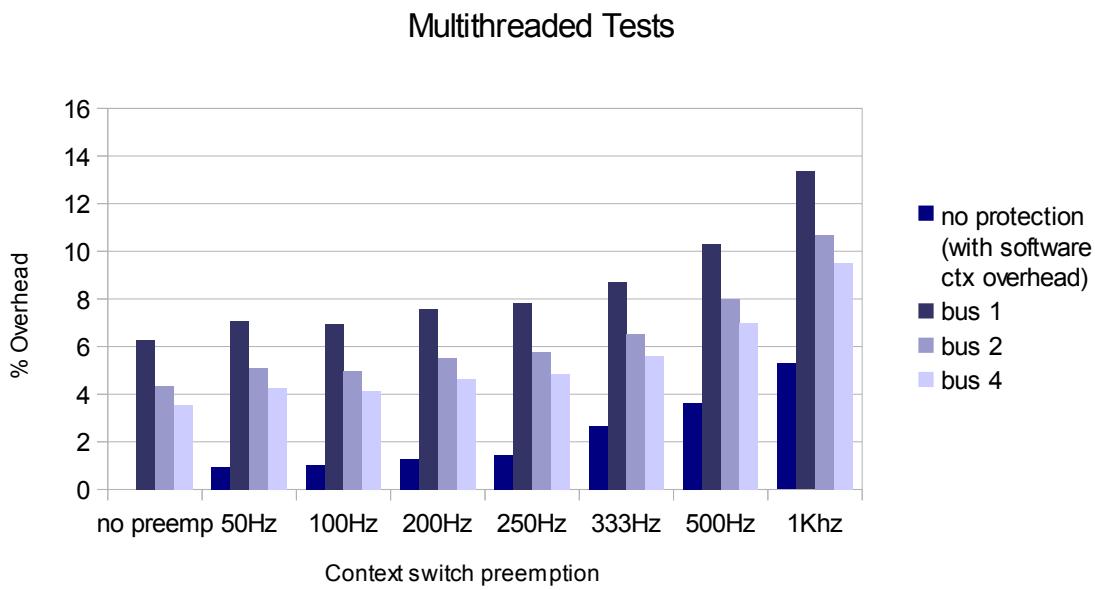


Figure 5. Context switch cost as frequency increases. Shows how bus widening reduces the cost, and how the cost compares to the cost without any wrapper management during the context switch.

Function Call Frequency Tests

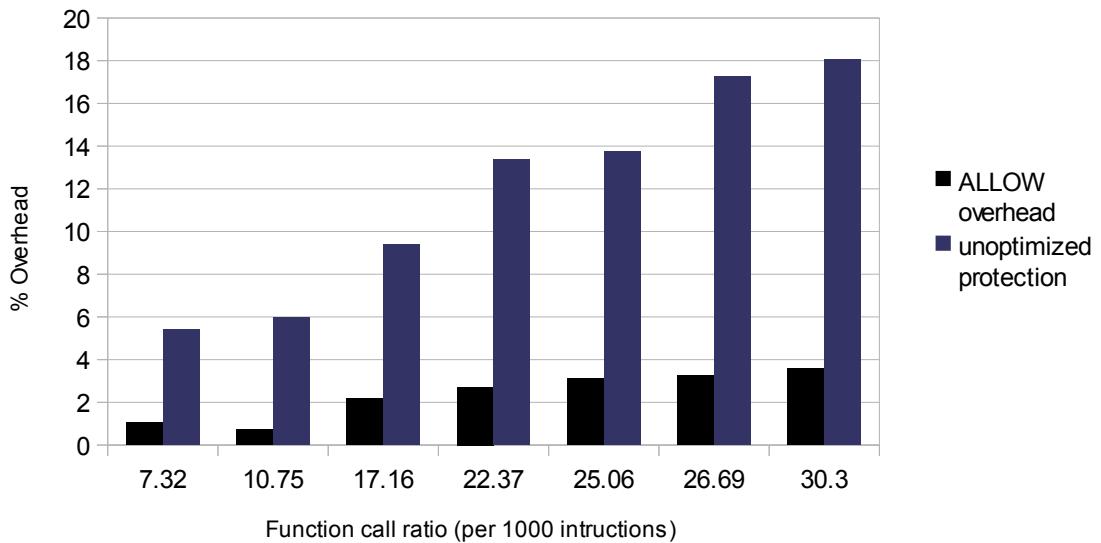


Figure 6. How varying the ratio of function calls to other instructions affects performance.